This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1. (Original) A method in a computer system for servicing requests from one or more client computers, the method comprising:

receiving a request from a client computer;

a first thread processing the request by invoking a receive handler that creates a work item, wherein the first thread is part of a pool of generic threads;

a second thread performing a task specified in the work item by invoking a work handler, wherein the second thread is part of the pool of generic threads;

receiving a result of performing the task; and

a third thread returning at least a portion of the result to the client computer by invoking a reply handler, wherein the third thread is part of the pool of generic threads.

2. (Original) The method as claimed in claim 1, wherein receiving the request comprises:

receiving the request into an input/output port; and

placing a reference in a queue indicating that work is available for the first thread.

3. (Original) The method as claimed in claim 2, further comprising: invoking a receive thread manager when the work is available; and

the receive thread manager scheduling the first thread for execution on one of multiple processors.

4. (Original) The method as claimed in claim 3, wherein the receive thread manager schedules the first thread from a queue of available threads.

5. (Original) The method as claimed in claim 1, further comprising the first thread placing the work item on a work queue for execution by the second thread.

6. (Original) The method as claimed in claim 5, further comprising the first thread placing a reference in a completion port queue, indicating that work is available for the second thread.

7. (Original) The method as claimed in claim 1, further comprising the second thread creating and placing a second work item on a reply work queue when the results are received.

8. (Original) The method as claimed in claim 7, further comprising the second thread placing a reference in a completion port queue, indicating that work is available for the third thread.

9. (Original) The method as claimed in claim 1, further comprising:

receiving input data; and

the first thread storing the input data in a cache that is accessible to the second thread.

10. (Original) The method as claimed in claim 1, wherein the results include data, and further comprising storing the data in a cache that is accessible to the third thread.

11. (Original) The method as claimed in claim 1, further comprising:

determining a size of the result; and

when the size of the result is not larger than a cutoff size, storing the result in the partial results cache.

12. (Original) The method as claimed in claim 11, further comprising:

the third thread returning a portion of the result to client computer; and

if a second request is received for an additional portion of the result that is stored in the partial results cache, a fourth thread creating a second work item by invoking another receive handler, wherein the second work item is processed by a fifth thread, which invokes another reply handler.

13. (Original) The method as claimed in claim 1, further comprising:

the first thread, the second thread or the third thread indicating that the first thread, the second thread or the third thread has completed a work item; and

if a quantum has not expired for the first thread, the second thread or the third thread, then the first thread, the second thread or the third thread being given an additional work item to perform without relinquishing the central processing unit upon which the first thread, the second thread or the third thread was running.

14. (Currently Amended) A method in a computer system for servicing requests from multiple client computers, the method comprising:

receiving a request from a client computer to perform a multi-state function;

performing a first task, by a first work handler invoked by a first thread in a ready state, wherein the first task is associated with a first state of the multi-state function, and performing the first task includes issuing an asynchronous request for data;

placing the first thread back in the ready state;

receiving the data specified in the asynchronous request; and

performing a second task, by a second work handler invoked by a second thread in the ready state, wherein the second task is associated with a second state of the multi-state function, and the second task performs an operation on the data, wherein the first thread and the second thread are all identical generic threads within a pool of generic threads.

15. (Original) The method as claimed in claim 14, further comprising:

processing the request by a receive handler invoked by a third thread;

creating a work item that specifies the first task; and

placing the work item in a work queue that is accessible to the first thread.

16. (Currently Amended) The method as claimed in claim 15, wherein the first thread, the second thread, and the third thread are all identical generic threads within ~~a~~ the pool of generic threads.

17. (Original) The method as claimed in claim 15, wherein placing the work item in the work queue comprises placing the work item in a low priority work queue, the method further comprising:

placing a second work item that specifies the second task on a high priority work queue;

a work handler looking for a work item first on the high priority work queue; and

if no work item exists on the high priority work queue, the work handler looking for the work item on the low priority queue.

18. (Original) The method as claimed in claim 14, wherein the asynchronous request is a request that would otherwise cause the first thread to block.

19. (Currently Amended) The method as claimed in claim 14[[8]], wherein the asynchronous request is a request that would otherwise cause a blocking condition to occur.

20. (Original) The method as claimed in claim 14, wherein issuing the asynchronous request comprises issuing the asynchronous request to a database manager.

21. (Original) The method as claimed in claim 20, further comprising:

the database manager placing the asynchronous request on a pending queue; and

when the data is received, placing a work item associated with the second state on a work queue.

22. (Original) The method as claimed in claim 21, wherein the work queue is a high priority work queue, the method further comprising:

placing a first work item that specifies the first task on a low priority work queue;

a work handler looking for a work item first on the high priority work queue; and

if no work item exists on the high priority work queue, the work handler looking for the work item on the low priority work queue.

23. (Original) The method as claimed in claim 14, further comprising returning a result of the first task and the second task by invoking third thread, which in turn invokes a reply handler to return the result to the client computer.

24. (Original) The method as claimed in claim 23, wherein the first thread, the second thread, and the third thread are all identical generic threads within a pool of generic threads.

25. (Original) The method as claimed in claim 14, further comprising:

performing additional tasks associated with subsequent function states by additional work handlers invoked by one or more additional threads;

at least some of the additional work handlers issuing additional asynchronous requests; and

placing threads associated with the at least some of the additional work handlers back in the ready state after issuing the additional asynchronous requests.

26. (Currently Amended) A method in a computer system for servicing requests from multiple client computers, the method comprising:

determining that work is available after receiving a request from a client computer, wherein the request from the client computer is a request to perform a function having multiple states;

when work is available, a first work handler invoked by a first thread looking in a first work queue for a first work item corresponding to the work, wherein the first thread is a generic thread within a pool of generic threads; and

if the first work item is not found in the first work queue, the first work handler looking in a second work queue for the first work item.

27. (Original) The method as claimed in claim 26, further comprising:

receiving a request from a client computer to perform a task;

creating the first work item that specifies the task;

placing the first work item in the second queue; and

indicating that the work is available.

28. (Original) The method as claimed in claim 26, further comprising:

the first work handler performing a task specified in the first work item; and

issuing an asynchronous request for data.

29. (Original) The method as claimed in claim 28, further comprising:

receiving the data;

placing a second work item on the first work queue; and

indicating that additional work is available.

30. (Currently Amended) The method as claimed in claim 29, further comprising:

a second work handler invoked by a second thread looking in the first work queue for the second work item when the second work is available, wherein the first and second threads are all identical threads within the pool of generic threads; and

performing a second task specified in the second work item.

31. (Original) The method as claimed in claim 26, wherein the computer system includes multiple work queues, including the first work queue and the second work queue, each of the multiple work queues are associated with a priority level, and wherein the method further comprises:

looking for the first work item first in a work queue associated with a highest priority level; and

if the first work item is not found in the work queue associated with the highest priority level, looking for the first work item in each of the multiple work queues in descending priority order until the first work item is found.

32. (Previously Presented) The method as claimed in claim 31, wherein each state of the multi-state function is performed by a work handler invoked by a subsequent thread based on work items placed in the multiple work queues, and wherein the work items are placed in higher and higher priority work queues as execution of the multi-state function progresses through the multiple states.

33. (Currently Amended) A method in a computer system for servicing requests from multiple client computers, the method comprising:

receiving, from a client computer, a request to perform a first task;

evaluating the first task, by a first handler invoked by a first thread, to determine whether the first task includes complex or long-running logic; and

if the first task includes complex or long-running logic, performing the first task by a second handler invoked by a second thread, wherein the first and second threads are all identical threads within a pool of generic threads.

34. (Original) The method as claimed in claim 33, wherein the first thread is a thread within a first group of threads having a first priority level, and the second thread is a thread within a second group of threads having a second priority level that is lower than the first priority level.

35. (Original) The method as claimed in claim 34, further comprising:

receiving a request to perform a second task;

evaluating the second task to determine whether the second task includes complex or long-running logic; and

if the second task does not include complex or long-running logic and a processor is not available for performing the second task, preempting the second thread and performing the second task by a third thread in the first group of threads.

36. (Original) The method as claimed in claim 33, further comprising:

the second handler returning a result of the first task; and

using the result, performing a second task by a third handler invoked by a third thread of the first group of threads.

37. (Original) The method as claimed in claim 36, wherein the first task is specified in a first work item and the second task is specified in a second work item, the method further comprising:

the first handler obtaining the first work item from a first work queue; and

the third handler obtaining the second work item from a second work queue.

38. (Currently Amended) A method in a computer system for servicing requests from one or more client computers, the method comprising:

maintaining a pool of threads, wherein each thread in the pool of threads is identical and can invoke at least one receive handler, at least one work handler, and at least one reply handler;

invoking receive handlers by the threads in response to receiving requests from one or more client computers, wherein the receive handlers create work items that specify tasks to be performed to satisfy the request;

invoking ~~invoicing~~ work handlers by the threads to perform the tasks specified in the work items;

receiving results of the tasks; and

invoking reply handlers by the threads to return at least portions of the results to the client computers.

39. (Original) The method as claimed in claim 38, wherein one or more of the receive handlers, one or more of the work handlers, and one or more of the reply handlers can be simultaneously executed on multiple processors available to the computer system.

40. (Original) The method as claimed in claim 38, wherein the pool of threads includes a number of threads in a range from about N+1 to about 2*N, where N is a number of processors available to the computer system.

41. (Original) A method in a computer system for servicing requests from multiple client computers, the method comprising:

monitoring a quantity of work being performed by the computer system;

determining whether the quantity has exceeded an upper limit; and

if the quantity has exceeded the upper limit but has not dropped below a lower limit, not accepting new requests into the computer system.

42. (Original) The method as claimed in claim 41, further comprising, if the quantity has exceeded the upper limit and has dropped below the lower limit, accepting the new requests into the computer system.

43. (Original) The method as claimed in claim 41, further comprising continuing to monitor the quantity of work being performed by the computer system after the quantity has exceeded the upper limit.

44. (Original) The method as claimed in claim 41, wherein the quantity of work is indicated by a number of work items on one or more work queues, and determining whether the quantity has exceeded the upper limit comprises determining whether the number of work items has exceeded a predefined value.

45. (Original) A method in a computer system for servicing requests from multiple client computers, the method comprising:

monitoring an amount of time to return a result by the computer system;

determining whether the amount of time has exceeded an upper limit; and

if the amount of time has exceeded the upper limit but has not dropped below a lower limit, not processing new work items.

46. (Original) The method as claimed in claim 45, further comprising, if the amount of time has exceeded the upper limit and has dropped below the lower limit, accepting and processing the new work items.

47. (Original) The method as claimed in claim 45, further comprising continuing to monitor the amount of time after the amount of time has exceeded the upper limit.

48. (Original) The method as claimed in claim 45, wherein the amount of time includes a time it takes to post the result to an output port, and determining whether the amount of time has exceeded the upper limit comprises determining whether the time it takes to post the result has exceeded a predefined value.

49. (Currently Amended) An application program for implementation by an application server, the application program comprising:

at least one ~~receiver~~ receive handler that can be invoked by a thread within a pool of threads, wherein each thread in the pool of threads is identical;

at least one work handler that also can be invoked by the thread; and

at least one reply handler that also can be invoked by the thread.

50. (Currently Amended) The application program as claimed in claim 49, wherein a receive~~r~~ handler of the at least one ~~receiver~~ receive handler is executed when the application server receives a request from a client computer, and the receive~~r~~ handler creates a first work item to be performed by a work handler of the at least one work handler.

51. (Original) The application program as claimed in claim 50, wherein the work handler is executed when the first work item exists, the work handler receives results, and the work handler creates a second work item to be performed by a reply handler of the at least one reply handler.

52. (Original) The application program as claimed in claim 51, wherein the reply handler is executed when the second work item exists, and the reply handler sends the results to the client computer.

53. (Original) The application program as claimed in claim 49, wherein the thread can invoke multiple work handlers, where some of the multiple work handlers are designed to perform tasks associated with various states of a multi-state function, and at least some of the multiple work handlers issue asynchronous requests for data when a state transition is to be performed, wherein the thread is then placed back in a ready state to execute a subsequent work item.

54. (Original) The application program as claimed in claim 53, wherein when the data is returned, a second work handler is executed.

55. (Original) The application program as claimed in claim 49, wherein multiple identical copies of the thread exist within the pool of threads, and at least one copy of the thread is executed each time a request is received from a client computer.

56. (Original) The application program as claimed in claim 49, wherein multiple copies of the thread can simultaneously be executed by multiple processors available to the application server.

57. (Original) The application program as claimed in claim 49, further comprising one or more complex logic handlers that can be invoked by a second type of thread, wherein a thread of the second type of thread is executed when a request from a client computer involves execution of complex or long-running logic.

58. (Original) A computer system for servicing requests from one or more client computers, the computer system comprising:

a memory containing an application server which maintains a pool of threads, wherein each thread in the pool of threads is identical and can invoke at least one receive handler, at least one work handler, and at least one reply handler, and the application server further schedules threads in the pool of threads for execution on multiple processors available to the computer system; and

the multiple processors for simultaneously executing multiple threads within the pool of threads.

59. (Original) The computer system as claimed in claim 58, wherein the application server further performs functions of:

receiving a request from a client computer;

a first thread processing the request by invoking a receive handler which creates a work item, wherein the first thread is part of the pool of threads;

a second thread performing a task specified in the work item by invoking a work handler, wherein the second thread is part of the pool of threads;

receiving a result of performing the task; and

a third thread returning at least a portion of the result to the client computer by invoking a reply handler, wherein the third thread is part of the pool of threads.

60. (Original) The computer system as claimed in claim 58, wherein the application server further performs functions of:

receiving a request from a client computer to perform a multi-state function;

a first thread within the pool of threads performing a first task by invoking a first work handler, wherein the first task is associated with a first state of the multi-state function, and performing the first task includes issuing an asynchronous request for data;

placing the first thread back in a ready state;

receiving the data specified in the asynchronous request; and

a second thread within the pool of threads performing a second task by invoking a second work handler, wherein the second task is associated with a second state of the multi-state function, and the second task performs an operation on the data.

61. (Original) The computer system as claimed in claim 58, wherein the application server further performs functions of:

determining that work is available after receiving a request from a client computer;

when work is available, a first thread within the pool of threads invoking a first work handler to look in a first work queue for a first work item corresponding to the work; and

if the first work item is not found in the first work queue, the first work handler looking in a second work queue for the first work item.

62. (Original) The computer system as claimed in claim 58, wherein the application server further performs functions of:

receiving, from a client computer, a request to perform a first task;

evaluating the first task, by a first handler invoked by a first thread within the pool of threads, to determine whether the first task includes complex or long-running logic; and

if the first task includes complex or long-running logic, performing the first task by a second handler invoked by a second thread that is not within the pool of threads.

63. (Original) The computer system as claimed in claim 58, wherein the application server performs functions of:

monitoring a quantity of work being performed by the computer system;

determining whether the quantity has exceeded an upper limit; and

if the quantity has exceeded the upper limit but has not dropped below a lower limit, not accepting new requests into the computer system.

64. (Original) The computer system as claimed in claim 58, wherein the application server further performs functions of:

monitoring an amount of time to return a result by the computer system;

determining whether the amount of time has exceeded an upper limit; and

if the amount of time has exceeded the upper limit but has not dropped below a lower limit, not processing new work items.

65. (Previously Presented) A computer-readable medium holding computer executable instructions, the computer-readable medium for performing a method in a computer system, the method comprising:

maintaining a pool of threads, wherein each thread in the pool of threads is identical and can invoke at least on receive handler, at least one work handler, and at least one reply handler, and the application server further schedules threads in the pool of threads for execution on multiple processors available to the computer system; and

simultaneously executing multiple threads within the pool of threads on multiple processors available to the computer system.

66. (Original) The computer-readable medium as claimed in claim 65, wherein the method further comprises:

receiving a request from a client computer;

a first thread processing the request by invoking a receive handler, which creates a work item, wherein the first thread is part of the pool of threads;

a second thread performing a task specified in the work item by invoking a work handler, wherein the second thread is part of the pool of threads;

receiving a result of performing the task; and

a third thread returning at least a portion of the result to the client computer by invoking a reply handler, wherein the third thread is part of the pool of threads.

67. (Original) The computer-readable medium as claimed in claim 65, wherein the method further comprises:

receiving a request from a client computer to perform a multi-state function;

a first thread within the pool of threads performing a first task by invoking a first work handler, wherein the first task is associated with a first state of the multi-state function, and performing the first task includes issuing an asynchronous request for data;

placing the first thread back in a ready state;

receiving the data specified in the asynchronous request; and

a second thread within the pool of threads performing a second task by invoking a second work handler, wherein the second task is associated with a second state of the multi-state function, and the second task performs an operation on the data.

68. (Original) The computer-readable medium as claimed in claim 65, wherein the method further comprises:

determining that work is available after receiving a request from a client computer;

when work is available, a first work handler invoked by a first thread within the pool of threads looking a first work queue for a first work item corresponding to the work; and

if the first work item is not found in the first work queue, the first work handler looking in a second work queue for the first work item.

69. (Original) The computer-readable medium as claimed in claim 65, wherein the method further comprises:

receiving, from a client computer, a request to perform a first task;

evaluating the first task, by a first handler invoked by a first thread within the pool of threads, to determine whether the first task includes complex or long-running logic; and

if the first task includes complex or long-running logic, performing the first task by a second handler invoked by a second thread that is not within the pool of threads.

70. (Original) The computer-readable medium as claimed in claim 65, wherein the method further comprises:

monitoring a quantity of work being performed by the computer system;

determining whether the quantity has exceeded an upper limit; and

if the quantity has exceeded an upper limit but has not dropped below a lower limit, not accepting new requests into the computer system.

71. (Original) The computer-readable medium as claimed in claim 65, wherein the method further comprises:

monitoring an amount of time to return a result by the computer system;

determining whether the amount of time has exceeded an upper limit; and

if the amount of time has exceeded the upper limit but has not dropped below a lower limit, not processing new work items.